PATENT APPLICATION

RETRIEVING DATA BLOCKS WITH REDUCED LINEAR ADDRESSES

INVENTOR(S): Stephan J. Jourdan
14664 NW Rich Court
Portland, OR 97229
Citizen of France

Chris E. Yunker
8951 SW 176th Avenue
Beaverton, OR 97007
Citizen of USA

Pierre Michaud
IRISA
Campus de Beaulieu
35042 RENNES Cedex
Citizen of France

ASSIGNEE: INTEL CORPORATION

KENYON & KENYON
1500 K Street, NW, Suite 700
Washington, DC 20005
Telephone: (202) 220-4200

Retrieving Data Blocks With Reduced Linear Addresses

BACKGROUND

Technical Field

[0001] Embodiments of the present invention generally relate to computers. More particularly, embodiments relate to retrieving data blocks in computer processing architectures.

Discussion

[0002] In the computer industry, the demand for higher processing speeds is well documented. While such a trend is highly desirable to computers, it presents a number of challenges to industry participants. A particular area of concern is data retrieval.

[0003] Modern day computer processors are organized into one or more "pipelines," where a pipeline is a sequence of functional units (or "stages") that processes instructions in several steps. Each functional unit takes inputs and produces outputs, which are stored in an output buffer associated with the stage. One stage's output buffer is typically the next stage's input buffer. Such an arrangement allows all of the stages to work in parallel and therefore yields greater throughput than if each instruction had to pass through the entire pipeline before the next instruction could enter the pipeline. In order to maximize the speed at which instructions are fed into the pipelines, data blocks including the instructions are organized into prediction arrays and various levels of cache, such as trace cache, instruction cache, etc. The prediction and cache architectures can be accessed relatively quickly and help reduce the need to access slower, off-chip memory.

[0004] When a full linear address of a data block is encountered, the data block is retrieved from a respective data array if the full linear address corresponds to a tag in a tag array, where the tag array is associated with the data array. If the data array is a prediction data array, the data block includes a branch prediction address having a size that equals the size of the full linear address. If the data array is a cache array, the data block includes, inter alia, a stored linear address having a size that equals the size of the full linear address. In either case, the tag array is indexed based on the full linear address, and therefore must be sized accordingly. As a result, fewer entries are available for the same sized data array, or a larger data array is required for the

same number of entries. Since more entries are known to enhance performance, a difficult tradeoff must often be made between size and performance. There is therefore a need for an approach to processing addresses that enables the use of smaller tag arrays and therefore larger data arrays within a fixed area budget (number of bits).

BRIEF DESCRIPTION OF THE DRAWINGS

[0005] The various advantages of embodiments of the present invention will become apparent to one skilled in the art by reading the following specification and appended claims, and by referencing the following drawings, in which:

[0006] FIG. 1 is a block diagram of an example of a front end architecture according to one embodiment of the invention;

[0007] FIG. 2 is a diagram of an example of a branch predictor that is accessed with a reduced linear address according to one embodiment of the invention;

[0008] FIG. 3 is a diagram of an example a cache that is accessed with a reduced linear address according to one embodiment of the invention;

[0009] FIG. 4 is a flowchart of an example of a method of processing addresses according to one embodiment of the invention.

[0010] FIG. 5 is a block diagram of an example of a processor pipeline according to one embodiment of the invention; and

[0011] FIG. 6 is a block diagram of an example of a computer system according to one embodiment of the invention.

DETAILED DESCRIPTION

[0012] FIG. 1 shows a front end architecture 10 in which reduced linear addresses are used to reduce the size of the tag arrays associated with branch predictor 12, front end cache level 0 (FC0) 14 and front end cache level 1 (FC1) 16 and/or to enhance the performance associated with the tag arrays by increasing the number of available entries. Architecture 10 has an instruction translation look-aside buffer (ITLB) 18 that contains next instruction pointer (IP) logic (not shown) and an address processing unit 44. When buffer 18 receives a full linear address from processor bus 24, a reduction module 46 reduces the size of the full linear address

to obtain a reduced linear address. In the illustrated example, the full linear address is thirty-two bits and the reduced linear address is twenty-three bits. Specific address sizes are provided to facilitate discussion only. Accordingly, larger or smaller values may be readily substituted for the values provided without parting from the spirit and scope of the embodiments of the invention.

[0013] A retrieval module 48 retrieves a data block from a branch predictor 12 prediction array if the reduced linear address corresponds to a tag in a tag array, where the tag array is associated with the prediction array. The reduced linear address enables the tag array of the branch predictor 12 to be smaller than in conventional approaches. The data block that is retrieved from the prediction array includes a branch prediction address having a size that equals the size of the reduced linear address. Thus, the branch prediction address is twenty-three bits long in the illustrated example.

[0014] In one approach, the cache arrays of FC0 14 and FC1 16 include a stored linear address having a size that equals the size of the full linear address. Thus, the cache arrays can be indexed with a smaller input barrier while the retrieved data block can be verified for accuracy. Alternatively, if the address reduction process is readily reversible, storage of the full linear address can be avoided by working backward from the branch prediction address for verification.

[0015] The illustrated FC0 14 is a trace cache, where allocation module 20 verifies that either the data block is consecutive with respect to a previous data block or the stored linear address corresponds to a calculated branch target address. Verification of whether the data block is consecutive can be performed by comparing the stored linear address, which can be viewed as a current linear instruction pointer (CLIP), with the next LIP (NLIP) from the preceding access to the trace cache. In this regard, the FC0 14 stores sequences of micro-operations, and functions essentially as a linked list. As a result, the data block retrieved from FC0 14 will also include an NLIP, which can be compared to the stored linear address of the next data block. Verification of whether the stored linear address corresponds to a calculated branch target address is performed when a misprediction has occurred and therefore the NLIP from the previous access is not valid.

[0016] It will also be appreciated that FC1 16 is illustrated as being an instruction cache where a decoder/branch address calculator (BAC) 22 decodes the data block and the allocation module 20 verifies that either the data block is consecutive with respect to a previous data block

or the stored linear address corresponds to a calculated branch target address. Bus 24 interconnects the buffer 18, the decoder 22 and the allocation module 20.

[0017] Turning now to FIG. 2, the relationship between a full linear address 26 and a reduced linear address 28 is shown in greater detail. Generally, a subset of the full linear address 26 is hashed to reduce the size of the full linear address 26. Specifically, the full linear address includes one or more line offset bits and one or more set index bits, where the set index bits and the offset bits are isolated from the hashing operation. Remaining bits 32 (32a, 32b) are hashed down to a reduced subset of bits 30. Thus, full linear address has a length of L1, whereas reduced linear address 28 has a length of L2.

[0018] The illustrated hashing operation is represented by H(22..12)=L(31..22)^L(21..12). It should be noted that bits zero through eleven are isolated from the hashing because they identify the data block 50 within data array 52. As a result, the indexing operation is faster. The reduced subset of bits 30, on the other hand, is compared to a tag 54 in tag array 56. The illustrated tag array 56 and data array 52 are part of a branch predictor such as branch predictor 12 (FIG. 1). Thus, the retrieved data block 50 includes a branch prediction address 58 that has a size that equals the size of the reduced linear address 28 (i.e., L2). A thread signature 33 can also be hashed with the remaining bits 32 to remove aliasing across threads that use the same linear address range. An example of such an approach can be represented by S(3..0)L(31..24)^L(23..12). In addition, the size of the reduced linear address 30 (i.e., L2) can be increased to further reduce aliasing. For example, adding a bit to the reduced linear address 28 doubles the number of addresses that can be represented and therefore halves the amount of aliasing present.

[0019] FIG. 3 illustrates the use of a reduced size branch prediction address 60 to retrieve a data block 62 from a data array 64, where the data block 62 includes a stored linear address 66. Branch prediction address 60 can be readily substituted for branch prediction address 58 (FIG. 2). Branch prediction address 60 has a reduced subset of bits 68, which are compared to the tag 70 in tag array 72 that corresponds to data block 62. It can be seen that the stored linear address 66 has a size that equals the size of the full linear address 26 (FIG. 2). Data array 64 and tag array 72 are part of a data structure of a cache. The cache may be a trace cache, instruction cache, etc.

[0020] FIG. 4 shows a method 34 of processing addresses. Method 34 can be implemented using any commercially available hardware and/or programming techniques. For example, method 34 can be implemented as a set of instructions capable of being executed by processor, where the instructions are stored in a machine-readable medium such as random access memory (RAM), read only memory (ROM), etc.

[0021] A full linear address is received at processing block 36 and the size of the full linear address is reduced at block 38 to obtain a reduced linear address. If it is determined at block 40 that the reduced linear address corresponds to a tag in a tag array, where the tag array is associated with a data array, a data block is retrieved from the data array at block 42. Block 43 provides for verifying the retrieved data block. Specifically, it can be verified that either the data block is consecutive with respect to a previous data block or the stored linear address corresponds to a calculated branch target address.

[0022] Turning now to FIG. 5, a processor 72 having one or more pipelines defined by stages instruction fetch (IF), instruction code (ID), execute (EX), memory (ME), and writeback (WB). The instruction fetch portion of the pipeline has a front end architecture 74 with an address processing unit 76. Front end architecture 10 (FIG. 1) can be readily substituted for front end architecture 74, and address processing unit 44 (FIG. 1) can be readily substituted for address processing unit 76.

[0023] FIG. 6 shows a computer system 78 having a system memory 80 such as a random access memory (RAM), read only memory (ROM), flash memory, etc., a system bus 82 coupled to the system memory 80 and a processor 84 coupled to the bus 82, where the processor 84 receives instructions from the system memory 80 and includes an address processing unit 86. The address processing unit 86 can be similar to address processing unit 44 (FIG. 1). While the illustrated computer system 78 retrieves the instructions from system memory 80, the instructions may also be retrieved from any appropriate "on chip" memory such as a trace cache, instruction cache, etc.

[0024] Those skilled in the art can appreciate from the foregoing description that the broad techniques of the embodiments of the present invention can be implemented in a variety of forms. For example, branch prediction can make use of bimodal, local, global and other techniques while benefiting from the principles described herein. Therefore, while the embodiments of this invention have been described in connection with particular examples

thereof, the true scope of the embodiments of the invention should not be so limited since other modifications will become apparent to the skilled practitioner upon a study of the drawings, specification, and following claims.